PATENT
5681-80400
P9307

"EXPRESS MAIL" MAILING LABEL NUMBER EL990142769US
DATE OF DEPOSIT JANUARY 29, 2004
I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 C.F.R. §1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, BOX PATENT APPLICATION, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450

Derrick Brown

SIMULTANEOUS EXECUTION OF TEST SUITES ON DIFFERENT PLATFORMS

By:

Victor Rosenman and Olga Kuturianu

B. Noël Kivlin
Meyertons, Hood, Kivlin, Kowert & Goetzel
P.O. Box 398
Austin, TX 78767-0398

Simultaneous Execution of Test Suites on Different Platforms

CROSS-REFERENCE TO RELATED APPLICATIONS

This Application relates to Application No. (STC File No. 47900), entitled "Automated Test Execution Framework with Central Management", and to Application No. (STC File No. 47979), entitled "Parallel Text Execution on Low-End Emulators and Devices".

BACKGROUND OF THE INVENTION

1. Field of the Invention.

[0001] This invention relates to improvements in software and hardware design verification. More particularly, this invention relates to methods and systems for centrally managing the composition and execution of test suites for verification of different hardware and software.

2. Description of the Related Art.

[0002] The meanings of acronyms and certain terminology used herein are given in Table 1:

Table 1

API	Application programming interface
CLDC	Connected, limited device configuration. CLDC is suitable for devices with 16/32-bit RISC/CISC microprocessors/controllers, having as little as 160 KB of total memory available.
HTTP	HyperText Transfer Protocol
ID	Identifier
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD	Java application descriptor
JAR	Java archive
JDTS	Java Device Test Suite execution framework

MIDlet	A MIDP application
MIDP	Mobile information device profile. A set of Java APIs, which, together with the CLDC, provides a complete J2ME application runtime environment targeted at mobile information devices.
Platform	An underlying system on which application programs can run on a computing device. It typically includes an operating system, and can include supporting hardware, which may be either standardized, or customized for a particular computing device.

[0003] MIDP is defined in Mobile Information Device Profile (JSR-37), JCP Specification, Java 2 Platform, Micro Edition, Ver 1.0a (Sun Microsystems Inc., Palo Alto, California, December 2000). MIDP builds on the Connected Limited Device Configuration (CLDC) of the Java 2 Platform, Micro Edition (J2ME) (available from Sun Microsystems Inc., Palo Alto, California). The terms Sun, Sun Microsystems, Java, J2EE, J2ME, J2SE, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States of America and other countries. All other company and product names may be trademarks of their respective companies. A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by any one of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

[0004] Tools have been developed in recent years to aid in the design verification of hardware and software systems, for example software suites, hardware circuitry, and programmable logic designs. In order to assure that the design complies with its specifications, it is common to generate a large number of input or instruction sequences to assure that the design operates as intended under a wide variety of circumstances. In

general, test systems produce a report indicating whether tests have been passed or failed, and, in some cases may even indicate a module that is estimated to be faulty.

[0005] Conventionally, to test a device under development (such as a mobile information device), or to test software designed to run on such a device, a developer connects the device to an appropriate test system. The target device under test may be connected to the test system either directly or via a communication emulator. The developer selects a battery of test programs to run on the target device while monitoring its behavior. Running the complete battery of tests can commonly take many hours or even days. This problem is particularly acute in testing low-end computing devices, such as cellular telephones and other mobile information devices, which have limited computing power and memory resources. Thus, testing on the target device can become a serious bottleneck in the development cycle.

SUMMARY OF THE INVENTION

[0006] A centralized system for centrally managing test suites is disclosed in commonly assigned Application No. (STC File No. 47900), entitled "Automated Test Execution Framework with Central Management", which is herein incorporated by reference. In this arrangement, a central repository contains a management unit, available test suites and a single test execution framework, referred to herein as a test harness. Using the management unit, a system administrator establishes active versions of the various test suites and their individual configurations. End users install clients of the central repository, using a system-provided installer program. In each client, an execution script is created, which downloads the test harness and a local configuration file. Then, when the test harness is

executed at the client, it loads with all designated test suites already installed, configured and ready for execution. The client always has the most current versions of all test suites. Advantageously, all necessary information is obtained from a single central location.

[0007] A further improvement in test suite management is disclosed in commonly assigned Application No. (STC File No. 47979), entitled "Parallel Text Execution on Low-End Emulators and Devices", which is herein incorporated by reference. While this arrangement facilitates testing large numbers of devices simultaneously, it does not address the problem of simultaneously testing software that is designed to execute on different platforms. Nor does it address the related problem of simultaneously applying test suites to different computing devices, which employ different platforms. In order to avoid replication of test suite development, test suites may be designed to execute on different platforms. However, each platform typically provides a unique environment. Furthermore, the different platforms may require the use of different communication protocols and API's. Therefore, it would be desirable if test suites could be simultaneously executed on different platforms using a common test harness.

[0008] Embodiments of the present invention provide methods and systems for parallel testing of multiple low-end computing devices (devices-under-test), such as mobile information devices in which different test suites are executed simultaneously on different platforms from a single instance of a test harness. The test harness has a component-based architecture. This is exploited according to the instant invention by the substitution of various software components that support a test suite in order to execute the test suite on a new platform or on a new computing device. For example, a platform-specific

API is provided for each of the platforms upon which the test suite is executed on the clients of the test harness, that is by the devices-under-test. Other platform-specific software components needed to support execution of tests, e.g., an execution agent, and subsidiary classes, are implemented for each test suite according to the requirements of its respective platform-specific API. At runtime, the test harness deploys each test suite together with a compatible platform-specific execution agent and other platform-specific components. The agents execute the test suites, and communicate with the test harness, returning test results according to a known API.

[0009] The invention provides a method for testing computing devices, which is carried out by providing a plurality of suites of test programs for access by a server, wherein a first suite and a second suite of the plurality of suites are respectively adapted to run on a first platform and a second platform. The method further includes storing a first execution agent that is adapted to run on the first platform and a second execution agent that is adapted to run on the second platform for access by the server, and coupling a first computing device and a second computing device of the computing devices to the server, wherein the first computing device is adapted to operate using the first platform and the second computing device is adapted to operate using the second platform. The method further includes installing no more than one test harness on the server to support execution of the test programs by the first computing device and the second computing device, using the test harness to package a first test object with the first execution agent for download to the first computing device in a first package and to package a second test object with the second execution agent for download to the second computing device in a second package. The method further includes steps of re-

sponsively to an instruction of the test harness, downloading the first package and the second package to the first computing device and the second computing device, respectively, and concurrently executing a test program of the first package in the first computing device and a test program of the second package in the second computing device.

[0010] According to an aspect of the method, the first suite and the second suite comprise platform-specific JAR files.

[0011] According to yet another aspect of the method, the first package and the first package comprise JAR files.

[0012] A further aspect of the method is carried out by displaying the suites as a hierarchy of identifiers of test objects corresponding to the test programs, and selecting the first test object from the first suite for execution thereof by the first computing device, and selecting the second test object from the second suite for execution thereof by the second computing device.

[0013] The invention provides a computer software product, including a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to perform a method for testing computing devices, which is carried out by providing a plurality of suites of test programs for access by a server, wherein a first suite and a second suite of the plurality of suites are respectively adapted to run on a first platform and a second platform. The method further includes storing a first execution agent that is adapted to run on the first platform and a second execution agent that is adapted to run on the second platform for access by the server, and coupling a first computing device and a second computing device of the computing devices to the server, wherein the first computing device is

adapted to operate using the first platform and the second computing device is adapted to operate using the second platform. The method further includes installing no more than one test harness on the server to support execution of the test programs by the first computing device and the second computing device, using the test harness to package a first test object with the first execution agent for download to the first computing device in a first package and to package a second test object with the second execution agent for download to the second computing device in a second package. The method further includes steps of responsively to an instruction of the test harness, downloading the first package and the second package to the first computing device and the second computing device, respectively, and concurrently executing a test program of the first package in the first computing device and a test program of the second package in the second computing device.

[0014] The invention provides a system for testing computing devices, including a communication interface for coupling a plurality of the computing devices thereto for use in communicating with the system via the communication interface, a memory, a single test harness object stored in the memory, a suite of test programs stored in the memory for execution by the computing devices that are coupled to the system, and a processor that accesses the suite and the test harness object, wherein the processor cooperates with the test harness object to download the test programs via the communication interface for execution by the computing devices coupled thereto, so that at least first and second computing devices among the plurality execute different first and second test programs from the suite. The system provides facilities to receive messages via the communication interface from the computing devices with respect to execution of the test programs, and to control the

execution of the test programs in the suite based on the messages by communicating responses to the messages via the communication interface, and wherein the first and second test programs are adapted to respective first and second platforms, and the first and second computing devices operate using the first and second platforms, respectively.

[0015] According to an aspect of the system, the first and second test programs are executed substantially simultaneously under control of the processor.

[0016] Still another aspect of the system the test harness object and the processor further cooperate to access first and second execution agents that are adapted to the first and second platforms, respectively, and to package the first and second test programs with the first and second execution agents, respectively, for download to the first and second computing devices.

[0017] Another aspect of the system includes a graphical user interface in the processor for displaying the test programs as a hierarchy for selection of the first and second test programs therefrom.

[0018] According to one aspect of the system, the computing devices are coupled to the communication interface via a common test host.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] For a better understanding of the present invention, reference is made to the detailed description of the invention, by way of example, which is to be read in conjunction with the following drawings, wherein like elements are given like reference numerals, and wherein:

[0020] Fig. 1 is a block diagram that schematically illustrate systems for parallel testing of low-end computing de-

vices, in accordance with an embodiment of the present invention;

[0021] Fig. 2 is a block diagram that schematically illustrate systems for parallel testing of low-end computing devices, in accordance with an alternate embodiment of the present invention;

[0022] Fig. 3 is a block diagram that schematically illustrates program components used in a test system, in accordance with an embodiment of the present invention;

[0023] Fig. 4 is a detailed block diagram of aspects of the program components shown in Fig. 3, in accordance with an embodiment of the invention; and

[0024] Fig. 5 is a flow chart illustrating a method for executing tests on a device that is attached to a test harness, in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0025] In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent to one skilled in the art, however, that the present invention may be practiced without these specific details. In other instances well-known circuits, control logic, and the details of computer program instructions for conventional algorithms and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

[0026] Software programming code, which embodies aspects of the present invention, is typically maintained in permanent storage, such as a computer readable medium. In a client/server environment, such software programming code may be stored on a client or a server. The software programming code may be embodied on any of a variety of known media for use with

a data processing system, This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, compact discs (CD's), digital video discs (DVD's), and computer instruction signals embodied in a transmission medium with or without a carrier wave upon which the signals are modulated. For example, the transmission medium may include a communications network, such as the Internet.

System Overview.

[0027] Reference is now made to Fig. 1, which is a block diagram that schematically illustrates a system 20 for parallel testing of multiple mobile information devices 24, in accordance with an embodiment of the present invention. The system 20 is built around a test server 22, which is described in greater detail hereinbelow. The devices 24 are client devices, and are typically low-end devices, with limited computing power and memory, for example, cellular telephones or personal digital assistants (PDA's). In the description that follows, the devices 24 are assumed to comply with MIDP, but the principles of the present invention are equally applicable to other types of low-end computing devices, operating in accordance with other standards and specifications. The server 22 typically comprises a programmable processor, and has suitable memory and communication interfaces, such as wireless or wired interfaces, for communicating with multiple devices 24 simultaneously.

[0028] Each of the devices 24 typically receives a unique identifier for communicating with the server 22, which can be assigned in accordance with the methods disclosed in the above-noted Application No. (STC File No. 47979).

[0029] Reference is now made to Fig. 2, which is a block diagram that schematically illustrates a system 30 for

parallel testing of multiple devices 24, in accordance with another embodiment of the present invention. In this embodiment, the server 22 communicates with the devices 24 through a test host 32, such as a personal computer or workstation. Multiple test hosts of this sort may be connected to the server 22 in parallel, but only a single host is shown in Fig. 2 for the sake of simplicity. The host 32 can simultaneously accommodate multiple devices 24.

[0030] Reference is now made to Fig. 3, which is a block diagram that schematically illustrates software program components running on the server 22 and the devices 24, in accordance with an embodiment of the present invention. Elements of this software may be provided to the server 22 and to the devices 24 on tangible media, such as optical or magnetic storage media or semiconductor memory chips. The software may be downloaded to the server 22, and alternatively or additionally, to the devices 24 in electronic form, for example, over a network or over the air.

[0031] The server 22 comprises a test harness 40, which generates and deploys the tests to be carried out by the devices 24. The test harness 40 is realized as one or more objects stored in the memory of the server 22. The test harness 40 may be implemented as a modification of the "Java Device Test Suite" execution framework (JDTS) (version 1.0 or higher), available from Sun Microsystems, Inc., which employs MIDP. The test harness described in the above-noted Application No. (STC File No. 47900) can be modified for use as the test harness 40 by the application of ordinary skill.

[0032] A test manager 42 in the server 22 is responsible for serving requests from the devices 24, based on the unique client identifiers mentioned above. Typically, whenever one of the devices 24 makes a request, the test manager 42,

typically operating as a main thread, reads the request and assigns a new thread 44 to handle it. This thread 44 retrieves the client unique identifier from the request, calls the components of the test harness 40 that are needed to process the request, and then returns the appropriate response to the client device, as described hereinbelow. After assigning the thread 44 to handle the client, the main thread of the test manager 42 waits for the next client request. Each client request is handled by a separate thread 44, which terminates upon completion of processing. This arrangement, together with the unique identifier mechanism, ensures that the server 22 is able to handle multiple devices 24 simultaneously without confusion.

[0033] In order to run Java applications, the devices 24 contain an implementation of the Connected Limited Device Configuration specification, CLDC 46, with an implementation of the Mobile Information Device Profile specification, MIDP 48, running over the CLDC 46. The applications that run on this technology, such as the tests supplied by test harness 40, are known as MIDlets. These applications are created by extending an API MIDlet class of the MIDP 48. Thus, each test bundle 52 is actually a MIDlet, packaged in the form of a JAD/JAR file pair. At least a portion of the test bundle 52 is typically downloaded to the devices 24 in a two-step process:

[0034] 1. The server 22 downloads the JAD file, which contains environment settings and some environment demands. Application Manager Software, AMS 50, which is typically a part of a browser built into the devices 24, evaluates the JAD file to ensure that the device is able to accept the MIDlet. For example, the JAD file for a given MIDlet may specify that the device must support MIDP version 2.0. If the device does not support this version, the AMS 50 will

reject the application download, and will save the time that would otherwise be consumed by downloading the much larger JAR file.

[0035] 2. After completing all the relevant checks, the AMS 50 reads from the JAD file the location of the corresponding JAR file on the server 22 and asks to download the JAR file to one or more of the devices 24. The JAR file contains all the relevant classes of the test bundle 52.

[0036] Once the JAR file for the relevant portion of the test bundle 52 is downloaded to one of the devices 24 and stored in the local device memory, the device is ready to run the tests of the test bundle 52. Every JAR file that the AMS 50 downloads to the devices 24 typically contains an agent 54, which is used to run the tests, in addition to classes corresponding to the tests themselves. To start test execution the AMS 50 runs the agent class. The agent 54 then addresses the server 22 in order to receive instructions regarding the next test to run (getNextTest) and to report test results (sendTestResult), typically using a protocol based on HTTP. Each test in the test bundle 52 corresponds to a respective class in the JAR file. Each client request that is addressed by the agent 54 to the server 22 includes the unique identifier that has been assigned to the particular one of the devices 24, so that the server 22 is able to recognize the client and serve it in the correct manner. Further details of the implementation of the server 22 are given in the above-noted Application No. (STC File No. 47979).

[0037] The test harness 40 has a component based architecture. An API is provided for independent components of the tests for each of the platforms upon which the test bundle 52 is executed at the devices 24. At the test harness 40, plat-

form-specific components are implemented for each test bundle 52 according to the respective platform-specific API. At runtime, the test harness 40 deploys each test bundle 52 together with the agent 54, which is platform-specific, and configured according to the components of the particular test bundle 52. In each of the devices 24, the agent 54 executes the test bundle 52, and communicates with the test harness 40, returning test results according to a common API.

Test Suite Selection.

[0038] Reference is now made to Fig. 4, which is a detailed block diagram of the test harness 40 (Fig. 3) in accordance with a disclosed embodiment of the invention. Each of the available test suites 42 is associated with one of a group of execution agent JAR files 44 and one of a group of test API JAR files 46. The agent JAR files 44 and the API JAR files 46 are platform-specific. They are accessed by the test harness 40 when packaging tests for download. The tests typically are packaged in the form of Java applications contained in a set of JAD and JAR files. Each JAR file of this sort, together with its accompanying JAD file, is referred to hereinbelow as a test bundle 52. As explained in further detail hereinbelow, users of the system 20 (Fig. 1) or the system 30 (Fig. 2) interact with the test harness 40 in order to select the tests to be executed by the system. Alternatively, other test harnesses may be used for generating the required test files, as will be apparent to those skilled in the art.

[0039] The test harness 40 is provided with a user interface 48 that permits visualization of all available test suites and selection of tests of the test suites to be run. As is explained in further detail hereinbelow, the user interface 48 displays the test suites and the tests comprising the

test suites as a hierarchy of test names or identifiers that reference test program objects.

[0040] Using a session component 50, which stores all the test suites, a tree is constructed by the test harness 40 that contains all the test suites, With the user interface 48 the tree is displayed for the user, who can choose any combination of tests from the display. The user selections are transmitted to the session component 50. The session component 50 relates to a test suites manager 52, which further interoperates with the test harness 40 as explained below.

Test Suite Deployment.

[0041] The test harness 40 includes a deployer 54, which receives tests from the test suites manager 52 that have been selected by the session component 50 from the test suites 42 for packaging into JAR files. In the deployer 54 each test is associated with one of test suites 42 that are held in the session component 50. The deployer 54 accesses the corresponding one of the agent JAR files 44 and the API JAR files 46 and creates a new JAR file that includes the test itself and its associated specific execution agent and test API. As noted above, one or more these new JAR files form a test bundle 52, which is downloaded to the devices 24 or to a separate test host, for example the test host 32 (Fig. 2). Alternatively, the download could be directed to an emulator (not shown) of the devices 24. An executor 56 initiates the download of the new JAR files to the devices 24.

A code fragment shown in Listing 1 presents further details of the current software implementation of the deployer 54. As explained above, the devices 24 are able to execute the tests as they employ the agent classes of the particular test bun-

dle 52, which are specific to the test suite and compatible with the different platforms being used by the devices 24.

[0042] Reference is now made to Fig. 5, which is a flow chart illustrating a method for executing tests on a device that is attached to a test harness in accordance with a disclosed embodiment of the invention. Only one device is illustrated for clarity of presentation, but it will be understood that different devices, operating on different platforms, can execute tests concurrently. The process steps are shown in a particular sequence in Fig. 5. However, it will be evident to those skilled in the art that many of them can be performed in parallel, asynchronously, or in different orders.

[0043] The process begins at initial step 58. Here a group of tests is selected for execution on a device-under-test. The device-under-test is connected to a test harness.

[0044] Next, at step 60, a current test from the group defined in initial step 58 is chosen.

[0045] Next, at step 62 a pre-defined test suite that includes the current test chosen at step 60 is associated with the current test.

[0046] Next, at step 64 an object is retrieved from a store corresponding to the test suite identified at step 62.

[0047] Next, at step 66 JAR files are extracted from the object that was retrieved in step 64. The JAR files are specific to the platform of the device-under-test.

[0048] Next, at step 68 classes required to operate the current test are extracted from the JAR files that were obtained in step 66. These classes include an execution agent, corresponding to the agent 54 (Fig. 3).

[0049] Next, at step 70 a new JAR file and an accompanying JAD file are packaged. The JAR file includes the classes extracted at step 68, including the execution agent.

[0050] Next, at step 72, the new JAD and JAR files that were prepared in step 70 are downloaded to the device-under-test for execution.

[0051] Control now proceeds to decision step 74, where it is determined if more tests remain to be processed. If the determination at decision step 74 is affirmative, then control returns to step 60.

[0052] If the determination at decision step 74 is negative, then control proceeds to final step 76 and the process terminates. However, the process may be immediately repeated so long as there are additional devices available for connection to the test harness. Such devices may operate on the same or on different platform as the first device. Thus, using the method illustrated in Fig. 5, any number of different devices may execute tests simultaneously, on different platforms, using only a single test harness.

[0053] It will be appreciated by persons skilled in the art that the present invention is not limited to what has been particularly shown and described hereinabove. Rather, the scope of the present invention includes both combinations and sub-combinations of the various features described hereinabove, as well as variations and modifications thereof that are not in the prior art, which would occur to persons skilled in the art upon reading the foregoing description.

COMPUTER PROGRAM LISTINGS

Listing 1

```
public class DeployerThread extends Thread {
/* Thread is a standard Java Class*/

private boolean deployerActive = false;
```

```
protected Iterator recordsIterator = null;
protected JarMaker jarMaker = null;
protected BundleDispatcher dispatcher = null;
protected String sep =
  System.getProperty("file.separator");
protected Hashtable hashOfInnerClasses = new Hashtable();
protected TestSuitesManager tsm = null;
protected String currentTS = "";
protected TestSuite testSuite = null;
private DefaultJarDeployer jarDeployer;
protected String delimeter = null;
private boolean flag = false;
private Environment props;
private FrameworkEnvironmentConfiguration config;

/**
 * this method gets list of agent (platform-specific)
 * jars required for the test suite, which is
 * currently executed
 */
protected Vector getAgentJarsList() {
//get controlling properties
boolean deployAgentClasses = true;
String deployAgentClassesString =
   verifyProperty("DeployAgentClasses");

if ((deployAgentClassesString != null) &&
 deployAgentClassesString.toLowerCase().equals("false"))
    deployAgentClasses = false;
//get platform-specific jars
else if (deployAgentClasses) {
    appendJar("AgentJar", agentJars);
    appendJar("TestAPIJar", agentJars);
}

return agentJars;
}


/**
 * Patch the deployer props that had JarMaker according to
 * the current test suite
 */
protected Properties getDeployerProps(){
Properties deploymentProps = jarMaker.getDeployerProps();
String agentProps =
  deploymentProps.getProperty("agentprops");

if(testSuite!=null){
```

```
    String agentJar =
      testSuite.getTestSuiteProperty("AgentJar");

  if(agentJar!=null){
    Properties defaultProps = new Properties();
    try
    {
      URL[] url = { new File(agentJar).toURL()};
      URLClassLoader loader = new URLClassLoader(url);
      InputStream is =
        loader.getResourceAsStream("agentclasses.map");

      if(is!=null){
        defaultProps.load(is);
      }
    }
    catch(Exception e)
    {
      e.getMessage();
      e.printStackTrace(System.err);
    }

  Enumeration keys = defaultProps.propertyNames();

  while(keys.hasMoreElements()) {
    String name = ((String)keys.nextElement()).trim();
    String value = defaultProps.getProperty(name);

    if (name.equals("RemoteCommunicator")){
      agentProps = checkAndAddProperty(agentProps,
        "communicator", value);
    }
    else if (name.equals("TestLauncher")){
      agentProps = checkAndAddProperty(agentProps,
        "TestLauncher", value);
    }
    else {
      throw new IllegalArgumentException
      ("agentclasses.map contains unexpected property.\n"
      +
      "should define communicator and launcher only");
    }
  }

  }

  deploymentProps.setProperty("agentprops", agentProps);
  }
```

```
return deploymentProps;
}

protected Properties getAgentProps
  (/*Properties deploymentProps,Properties modeProps*/) {
Properties deploymentProps =
  jarDeployer.getDeploymentProps();

Properties modeProps = props.getModeProps();
String agentNames =
 config.getFrameworkProperty("AgentProperties");
StringTokenizer st = new StringTokenizer(agentNames,",");
String agentProps = "";

while(st.hasMoreTokens()) {
  String propertyName = ((String)st.nextToken()).trim();
  String name =
   propertyName.substring(0,propertyName.indexOf("=") + 1);

  agentProps += name +
    deploymentProps.getProperty
    (propertyName.substring(propertyName.indexOf("=") + 1,
  propertyName.length())) + "\n";
}


if(agentProps.endsWith("\n"))
  agentProps =
    agentProps.substring(0,agentProps.lastIndexOf("\n"));

if(!props.getExecutionMode().equals("Standard")) {
  Enumeration keys = modeProps.propertyNames();
  while(keys.hasMoreElements()) {
    String key = (String)keys.nextElement()
        agentProps +="\n" + key + "=" +
        modeProps.getProperty(key);
  }

  if(agentProps.endsWith("\n"))
    agentProps =
     agentProps.substring(0,agentProps.lastIndexOf("\n"));
  }

  if(props.getExecutionMode().equals("Standard") &&
   !props.getSubMode().equals(""))

    agentProps+="\nScreenAnalyzer=" +
     deploymentProps.getProperty("ScreenAnalyzer");
```

```
    agentProps+="\nTestBeansServerURL=http://" +
      verifyProperty("ServerName") + ":" +
      verifyProperty("ServerPort") + "/test";

    if(testSuite!=null){
     ComponentManager.getComponent
      ("TestManager")).getTestSuitesManager();
      String agentJar =
        testSuite.getTestSuiteProperty("AgentJar");
      if(agentJar!=null){
        Properties defaultProps = new Properties();
      try{
        URL[] url = { new File(agentJar).toURL()};
        URLClassLoader loader =
          new URLClassLoader(url, null);

        InputStream is =
          loader.getResourceAsStream("agentclasses.map");
        if(is!=null){
          defaultProps.load(is);
        }
      }
      catch(Exception e){
        e.getMessage();
        e.printStackTrace(System.err);
      }

      Enumeration keys = defaultProps.propertyNames();
      while(keys.hasMoreElements()) {
        String name =
         ((String)keys.nextElement()).trim();
        String value = defaultProps.getProperty(name);
      if (name.equals("RemoteCommunicator")){
        agentProps = checkAndAddProperty(agentProps,
          "communicator", value);
      }
      else if (name.equals("TestLauncher")){
        agentProps = checkAndAddProperty(agentProps,
          "TestLauncher", value);
      }
    else{
      throw new IllegalArgumentException
        ("agentclasses.map contains unexpected property.
         \n" +
        "should define communicator and launcher only");
    }

    }
  }
```

```
}

return deploymentProps;
}

protected void appendJar
  (String newJarName, Vector jars) {
String newJar =
   testSuite.getTestSuiteProperty(newJarName);

if(newJar==null)
  newJar = props.getProperty(newJarName);

if ((newJar == null) || (!newJar.endsWith(".jar")))
  System.out.println(newJarName +
   " will not be included into deployed jars." +
   " If romized - IGNORE");
else {
  newJar = newJar.trim();

if (!jars.contains(newJar))
  jars.add(newJar);
}
}

/** Verifies correctness of the required property
 * Can be used by extending classes
 */
protected String verifyProperty(String propName) {
String propValue = props.getProperty(propName);
if ((propValue == null) || (propValue.length() == 0)) {
  /* if (propName.equals("ServerName")){
  try{
    return InetAddress.getLocalHost().getHostName();
  }catch(UnknownHostException uhe){
    uhe.printStackTrace();
  }
}*/

System.out.println(propName + " property not defined");
  return null;
}
  return propValue.trim();
}

/** Does the deployment work
 */
public void run() {
deployerActive = true;
```

```
try {
  //prepare deployement
  jarMaker.prepareDeployment();

  try {
  //list tests for deployment

  while(deployerActive && recordsIterator.hasNext()) {
    TestRecord testRecord = (TestRecord)
     recordsIterator.next();
    //no more TestRecords available - terminate
    // deployment
    if(testRecord==null) {
      deployerActive=false;
      return;
    }

    //Check test suite
    String tsName =
     testRecord.getValue(testRecord.TestSuite);
    if (tsName == null)
      throw new NullPointerException
("No test suite name provided to the Deployer");

    if (!currentTS.equals(tsName)) {
      //test suite changed
      //update test suite specific values
      currentTS = tsName;
      testSuite = tsm.getTestSuite(tsName);

      //jarMaker should be updated with thw agent jars
      //according to the current testsuite
      jarMaker.setAgentJars(getAgentJarsList());

      //it also should be updated with new properties
      //according to the current test suite
      //jarMaker.setDeployerProps(getDeployerProps());
      jarMaker.setDeployerProps(getAgentProps());
      flag=true;

      if (testSuite == null)
        throw new NullPointerException
("There is no TestSuite object for the test suite: "
       + tsName);
       //close jars and start continue
       // deployement from a new jar
       //this is to allow only 1 test suite in jar
       jarMaker.terminateDeployment();
```

```
          //closesand deploys currently open jar
          jarMaker.prepareDeployment();
          //marks a start for the further deployment
        }

        //dispatch test record
        TestEntry te = dispatch(testRecord, testSuite);

        //deploy jar files
        Maker.deployEntry(te, testSuite);
      }
      } catch (ConcurrentModificationException e) {
        System.out.println
          ("Deployment abnormally terminated");
        e.getMessage();
        e.printStackTrace();
        deployerActive = false;
        return;
      }

      //terminate deployment
      jarMaker.terminateDeployment();
      System.gc();
      System.out.println("free memory: " +
        Runtime.getRuntime().freeMemory());
      System.out.println("used memory: " +
        (Runtime.getRuntime().totalMemory() -
        Runtime.getRuntime().freeMemory()));
    }
    catch (IOException ie) {
      ie.printStackTrace();
      System.out.println("Deployment failed");
      return;
    }

    //test deployment completed
    dispatcher.deploymentCompleted();
    //add notification to the server, so in serial mode,
    //the server could start the getNextApp request.
    jarDeployer.notifyServerListener();
    System.out.println("Deployment completed");
    deployerActive = false;
  }

  /** Creates TestEntry corresponding to this test record
   * and appends it to the vector of test entries
   */
  protected TestEntry dispatch(TestRecord testRecord,
   TestSuite ts) {
```

```
//list all files required to be deployed
ArrayList allFilesList = new ArrayList();
listAllFiles(allFilesList, testRecord,
ts.getTestSuiteProperty("TestClassesDir"));

// creating TestEntry and prepare jar files list for
// deployment
String[] allFiles = new String[allFilesList.size()];
Iterator iter = allFilesList.iterator();
for(int i = 0; iter.hasNext(); i++)
{
  allFiles[i] = (String) iter.next();
  if(allFiles[i].startsWith(File.separator))
    allFiles[i]=allFiles[i].
     substring(1,allFiles[i].length());
}

allFilesList = null;
//update vector of TestEntries
String testClass =
  testRecord.getValue(TestRecord.TestPackage).
  replace('/','.').replace('\\','.')
  + "." + testRecord.getValue(TestRecord.TestClass);

if(testClass.startsWith("."))
  testClass =
   testClass.substring(1,testClass.length());

  //add test suite properties
  Properties tsProps =
   testSuite.getTestSuiteProperties();
  Properties testProps =
   getProperties(testRecord.getValue
   (TestRecord.TestProperties));
  testProps.putAll(tsProps);
  //Remove jad and manifest properties from test
  //properties list
  //The following syntax expected:
  //<jad>.1=JAD PROPERTY
  //<jad>.2=JAD PROPERTY
  //etc.
  Vector jadPropsVector = new Vector();
  Vector manifestPropsVector = new Vector();
  for (Enumeration e = testProps.propertyNames();
   e.hasMoreElements() ;) {
    String tp = (String) e.nextElement();
    if (tp.startsWith("<jad>.")) {
      jadPropsVector.add(testProps.getProperty(tp));
      //add the property value to the list of jad
```

```
        // properties
        testProps.remove(tp);
        //remove jad property from the test properties
        //list
      }
    else if (tp.startsWith("<manifest>.")) {
        manifestPropsVector.add
          (testProps.getProperty(tp));
        //add the property value to the list of jar
        //manifest properties
        testProps.remove(tp);
        //remove jad property from the test properties
        //list
      }
  }

  //this test is needed since jadProps has to be null if
  //   empty (according to internal convention)
  String[] jadProps = null;

  if (!jadPropsVector.isEmpty()) {
    int size = jadPropsVector.size();
    jadProps = new String[size];
    for (int i = 0; i < size; i++)
      jadProps[i] = (String) jadPropsVector.elementAt(i);
  }

  //this test is needed since manifestProps has to be
  // null if empty (according to internal convention)
  String[] manifestProps = null;
  if (!manifestPropsVector.isEmpty()) {
    int size = manifestPropsVector.size();
    manifestProps = new String[size];
    for (int i = 0; i < size; i++)
        manifestProps[i] = (String)
    manifestPropsVector.elementAt(i);
  }

  String propsString = constructPropsString(testProps);
  return new
  TestEntry(testRecord.getValue(TestRecord.TestName),
    testClass, propsString, allFiles, jadProps,
    manifestProps);
  }

  private Properties getProperties(String testProps) {
  Properties props = new Properties();
  if(testProps==null)
    return props;
```

```
TBStringTokenizer st =
  new TBStringTokenizer(testProps, delimeter);
while(st.hasMoreTokens()) {
  String token = st.nextToken();
  try {
    String name =
     token.substring(0,token.indexOf("="));
    String value = token.substring(token.indexOf("=")
     + 1,token.length());
    // index+1 to skip the '=' sign
    props.setProperty(name,value);
  }
  catch (StringIndexOutOfBoundsException e) {
    e.printStackTrace();
    System.out.println(token + " property ignored");
  }
}
return props;
}

private String constructPropsString
  (Properties testProps)
{
if (testProps == null) return "";

String propsList = null;
for (Enumeration e = testProps.propertyNames();
  e.hasMoreElements() ;) {
    String pName = (String) e.nextElement();
    String prop = pName + '=' +
      testProps.getProperty(pName);
    if (propsList == null)
      propsList = prop;
    else
      propsList += ',' + prop;
}

return propsList;
}

/** Searches all required files for this record,
 *  including its own class
 */
protected void listAllFiles(ArrayList requiredFiles,
  TestRecord testRecord,
  String TestClassesDir) {
// adding required files and inner classes (of test
// class and of required classes)
```

```
String required =
  testRecord.getValue(TestRecord.RequiredClasses);
if ((required != null) &&
  (required.trim().length() != 0)) {

  StringTokenizer ct =
   new StringTokenizer(required, ",");
  while (ct.hasMoreTokens()) {
    String reqPath, reqFile;
    String requiredFile = ct.nextToken().replace
      ('/', sep.charAt(0));

    //required class is in a different package from
    //test class
    int sepPos = requiredFile.lastIndexOf(sep);
    if (sepPos != -1) {
      reqPath = requiredFile.substring(0, sepPos);
      reqFile =
       requiredFile.substring(requiredFile.lastIndexOf(sep)
       + 1);
    }
    else {
      reqPath = "";
      reqFile = requiredFile;
    }
    requiredFiles.add(requiredFile);

    if (requiredFile.endsWith(".class"))
    //search inner classes of required class
          searchInnerClasses(reqPath,
    reqFile.substring(reqFile.lastIndexOf(sep) + 1,
    reqFile.indexOf(".class")),
    requiredFiles, TestClassesDir);
  }
}

//adding test class and it's inner classes
String testClassPackage =
  testRecord.getValue(TestRecord.TestPackage).replace('.',
sep.charAt(0));
requiredFiles.add(testClassPackage + sep +
  testRecord.getValue(TestRecord.TestClass) + ".class");
searchInnerClasses(testClassPackage,
  testRecord.getValue(TestRecord.TestClass),
  requiredFiles, TestClassesDir);
}

/**
* Looks for inner classes for the specified class
```

```
*/
protected void searchInnerClasses(String testPackage,
String testClass,
ArrayList requiredFiles,
String TestClassesDir) {
try {
  ArrayList innerClasses = null;


  // look for inner classes in package
  if (!hashOfInnerClasses.containsKey(testPackage)) {
  // check for inner classes in new package and add them to
  //hash
    File f = null;
    if (TestClassesDir.length() > 0) f = new
      File(TestClassesDir + sep +
      testPackage.replace('.',sep.charAt(0)));
    else f = new File(testPackage);

    String[] foundFiles = f.list();

    //put found inner classes into the hashtable
    if (foundFiles != null)
      for (int i=0; i < foundFiles.length; i++)
        if (foundFiles[i].endsWith(".class") &&
        (foundFiles[i].indexOf('$')!=-1)) {
          if (innerClasses == null)
          innerClasses = new ArrayList();
          innerClasses.add(foundFiles[i]);
        }
    if (innerClasses == null) innerClasses = new
      ArrayList(0);
    hashOfInnerClasses.put(testPackage, innerClasses);
  }
  else
    innerClasses = (ArrayList)
      hashOfInnerClasses.get(testPackage);
    //look for class inner classes
    Iterator iter = innerClasses.iterator();
    while (iter.hasNext()) {
      String currentInner = (String) iter.next();
      if (currentInner.startsWith(testClass) &&
       (currentInner.indexOf("$") != -1))
        requiredFiles.add(testPackage + sep +
        currentInner);
    }
  }
catch (Exception e) {
  e.printStackTrace();
```

```
      System.out.println("Failed to search inner classes for
      class: " + testClass + " in package: " + testPackage);
    System.out.println
      ("Absence of inner classes might cause test execution
      errors");
  }
  }

  protected String checkAndAddProperty(String agentProps,
  String field, String value){
  int fieldIndex = agentProps.indexOf(field);
  if (fieldIndex != -1){
    String firstPart = agentProps.substring(0, fieldIndex +
      field.length() +1);
    String secontPart = agentProps.substring(fieldIndex +
       field.length() +1);
    int newlineIndex = secontPart.indexOf((int)'\n');
    String newValueString = value +
      secontPart.substring(newlineIndex) + "\n";
    agentProps = firstPart + newValueString;
  }
  return agentProps;
  }
  }
```